

# Java™ magazine

By and for the Java community



## GROWING ON OPEN

Java provides  
open source  
foundation for  
**AgroSense** and  
**Hadoop**

14

THE ECLIPSE  
FOUNDATION'S  
MIKE MILINKOVICH

38

JAVAFX IN  
SPRING

## COMMUNITY

02

### From the Editor

04

### Java Nation

News from JavaOne, plus people, events, and books

## JAVA TECH

30

New to Java

### Can You Teach Testing to Beginners?

Michael Kölling wraps up his series on BlueJ with a look at its interactive-testing support.

34

Java Architect

### Exploring Lambda Expressions for the Java Language and the JVM

Ben Evans, Martijn Verburg, and Trisha Gee help you get ready for lambda expressions in Java SE 8.

42

Mobile and Embedded

### Wirelessly Recover Your Device's Address Book

Vikram Goyal explains how to add a recovery model to the backup capability of your device's address book.

46

Mobile and Embedded

### Get Started with Java SE for Embedded Devices on Raspberry Pi

Bill Courington and Gary Collins walk you through getting Linux and Java SE for Embedded Devices to run on the Raspberry Pi in less than an hour.

55

Polyglot Programmer

### Building Actor-Based Systems Using the Akka Framework

Ted Neward shows you how to use Akka's open source actor model implementation to build distributed systems.

60

### Fix This

Take our JDBC code challenge.



14

JCP Executive Series

### Q&A WITH MIKE MILINKOVICH

The Eclipse Foundation's executive director assesses the state of Java and the JCP.

20

Java in Action

## GROWING ON OPEN

AgroSense provides an all-Java open source platform for sustainable farming and precision agriculture.

25

Java in Action

### AN ENGINE FOR BIG DATA

Hadoop uses Java for large-scale analytics.

38

Rich Client

### JAVAFX IN SPRING



Stephen Chin shows you why to use the Spring framework on the client.

A woman with blonde hair is sitting at a desk, looking at a computer monitor. She is wearing a dark top. The background shows a window with blinds and some office equipment. In the bottom left corner, there is a small icon of a play button inside a square frame.



**//send us your feedback /**

We'll review all suggestions for future improvements. Depending on volume, some messages may not get a direct reply.





PHOTOGRAPH BY BOB ADLER

02


[blog](#)
[Java .net](#)




[ABOUT US](#)
[JAVA TECH](#)
[JAVA IN ACTION](#)

COMMUNITY



# Answers. Fast.

Subscribe and gain instant cutting-edge development insight from the world's most trusted tech publishers — all in one on-demand digital library.

- Get unlimited access to 23,000+ online books and training videos from 100+ publishers
- Search full text to efficiently get the answers you need
- Make notes and organize content into folders you control
- Elevate your career with thousands of top business and professional development titles



**ANYTIME, ANYWHERE:**



Access on your desktop, notebook, tablet or mobile device.

**Start a free unlimited-access trial today** [safaribooksonline.com/javamag](http://safaribooksonline.com/javamag)

# MAKE THE FUTURE JAVA

**The theme of JavaOne 2012, held September 30–October 4, was “Make the Future Java.”**

Throughout the week, attendees explored Java's continued role as the most popular, complete, productive, secure, and innovative development platform as well as its open, transparent, collaborative, and community-driven evolution. At Sunday's Strategy keynote, Oracle's **Hasan Rizvi** detailed the three factors critical to Java's success: technology innovation, community participation, and Oracle's leadership/stewardship. Under technology, he noted Macintosh OS X and Linux ARM support on Java SE, open sourcing of JavaFX by the end of 2012, the release of Oracle Java Embedded Suite 7.0, and multiple releases on the Java EE side. Under community, he said that the Java Community Process (JCP) continues, with new JSR activity and Java user group participation up 25 percent since last year.

PHOTOGRAPHS BY HARTMANN STUDIOS

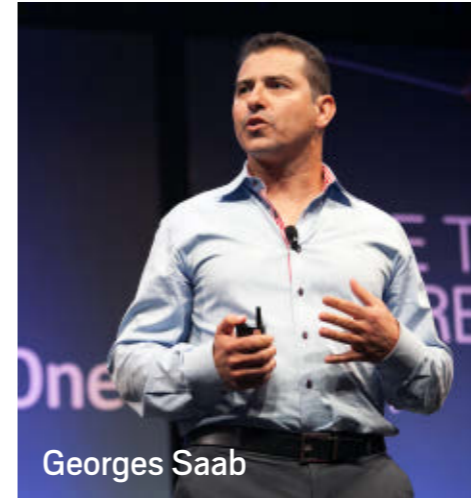


# Brian Goetz

Under stewardship, he noted Oracle's continued outreach—with four regional JavaOne conferences last year and the launch of *Java Magazine*. Here are highlights from Sunday's Strategy and Technical keynotes:

## Java 8/Java 9

Oracle's **Georges Saab** discussed the upcoming JDK 8 release—including Project Lambda and Project Nashorn (a modern implementation of JavaScript running on the Java Virtual Machine). He noted that Nashorn functionality has already been used internally in NetBeans 7.3, and



## Georges Saab



Mark Reinhold



**Nandini Ramani**

announced that Oracle plans to contribute the implementation to OpenJDK.

Oracle's **Brian Goetz** explored language and library features planned for Java SE 8, including lambda expressions and better parallel libraries. These feature changes both simplify code and free up libraries to more effectively use parallelism.

Oracle's **Mark Reinhold** urged developers to get involved in the Java 8 development process—getting the weekly builds, trying out their current code, and trying out the new features.



# JAVA IN ACTION



○

f

ava  
net

olog



06

06

06



Following the tradition of [JavaOne Latin America 2011](#), a Geek Bike Ride marked the beginning of JavaOne 2012 on a gorgeous day in San Francisco. About 50 Java developers gathered at Fisherman's Wharf, rode or skated through Fort Mason and Crissy Field, crossed the Golden Gate Bridge, and finished in Sausalito. Riders returned to San Francisco by ferry after a well-earned lunch. Participants donned Duke bike jerseys, sponsored by Oracle Technology Network. To receive a jersey, participants had to answer a question about Java. Questions included "Who is the father of Java?" "What's the biggest Java conference in San Francisco?" and "Name one Duke's Choice Award winner from this year," to which participant **Régina ten Bruggencate** answered, "Me!"



Clockwise from top: Geek Bike Ride participants paused for a photo op at the water's edge; SouJava's Juggy joined in on the ride; bikers enjoyed a leisurely pace.

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// NOVEMBER/DECEMBER 2012

# JAVA IN ACTION

JAVA TECH

## ABOUT US

○

## f

ava  
net

olog



09

09

09

09









**JavaOne Latin America** *DECEMBER 4-6,  
SÃO PAULO, BRAZIL*

Join Java developers and technologists for Java-focused content, training, and networking. Experts from the worldwide Java community share unique and leading-edge content with attendees. As always, there are keynotes, technical sessions, hands-on labs, demos, exhibitors, and more. Featured content tracks include Core Java Platform; Development Tools and Techniques; Emerging Languages on the JVM; Enterprise Service Architectures and the Cloud; Java EE Web Profile and Platform Technologies; Java ME, Java Card, Embedded, and Devices; and JavaFX and Rich User Experiences. And of course, there are opportunities to network so that you can build your community, share your expertise, and learn best practices.

DECEMBER 3-5  
GOTHENBURG, SWEDEN

JDays 2012 is a conference about Java, open source, and related technologies. The first two days of the conference feature a lineup of sessions that were voted on by the community. Day 3 of the conference offers a full day of free courses and hands-on training.

DECEMBER 8  
BANGALORE, INDIA

The Java Conference is an ideal place to obtain critical skills to help you build, run, and manage tomorrow's software solutions. The speakers are industry practitioners, selected for their knowledge and their ability to

convey that knowledge to Java developers at large. Spend time with hundreds of other Java professionals, share best practices, and take home practical advice that will make an immediate and measurable difference to your projects.

DECEMBER 13-14  
LONDON, ENGLAND

This two-day conference features two dozen expert-led talks, along with discussion and brainstorming sessions, all focused on learning and sharing ideas, tools, and best practices for enterprise Web development with Groovy and Grails.

DECEMBER 13-15  
PUNE, INDIA

This conference explores a wide array of software development tools and technologies in Java; cloud computing; mobile application development; and emerging technologies such as big data, gamification, HTML5, and more.


JANUARY 17-18  
LILLE, FRANCE

This English-language conference for Web developers and designers focuses on trending topics. Session content ranges from server side to pure front end, from design and philosophy to new languages, and from frameworks to development techniques.

FEBRUARY 4-6  
STOCKHOLM, SWEDEN

Jfokus is the largest annual conference for everyone who works with Java in Sweden. It is arranged in collaboration with Javaforum Stockholm, a Swedish developer community and a Java user group. Over three days, get up to date on the latest developments in the Java platform. The agenda includes rock-star speakers, both from Sweden and around the world, with a focus on systems development with Java and surrounding techniques such as dynamic languages and agile methodologies. Get the latest trends and buzz about Java from people who live and breathe technology.

(open source )

A man with short grey hair and glasses is seated, facing the camera. He is wearing a black zip-up hoodie with the Eclipse logo on the left chest. He is gesturing with both hands, palms facing up, as if explaining something. The background is a bright, out-of-focus window. In the foreground, the back of a person's head and shoulders are visible, slightly out of focus.

**C**ontinuing our series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP), we turn to Mike Milinkovich, executive director of the Eclipse Foundation, which was created in January 2004 as an independent not-for-profit corporation to foster a vendor-neutral, open, and transparent Eclipse community. Historically, Eclipse became famous as a Java IDE and a plug-in-based platform for building software development tools.

COMMUNITY

JAVA IN ACTION

JAVA TECH

ABOUT US

0

f

bird

# A CONVERSATION WITH MIKE MILINKOVICH

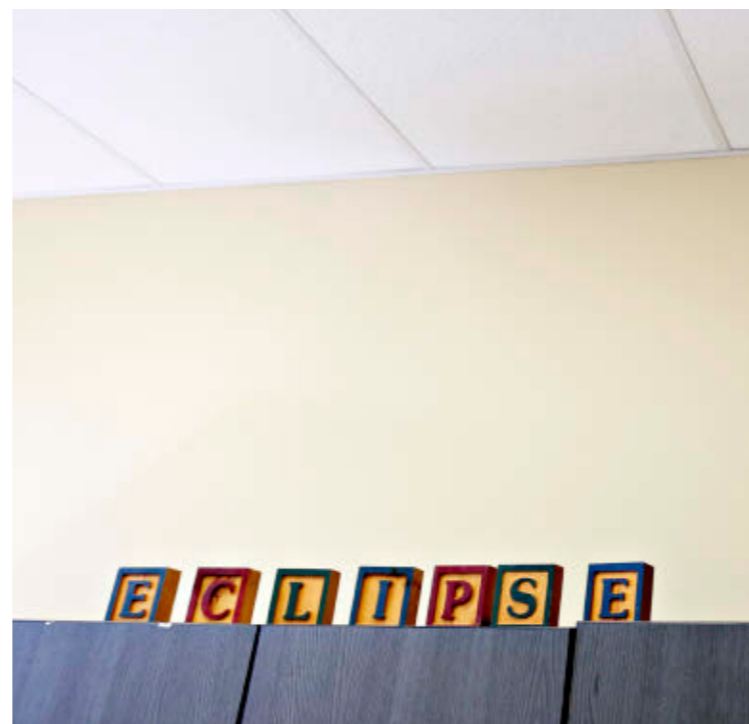
ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY BLAIR GABLE/GETTY IMAGES







**Left: Milinkovich analyzes some development details with Wayne Beaton, director of open source projects at Eclipse. Right: Spelling things out.**



things going on. So there are areas where I think they get it right and some where it's not so good.

First, innovating via standards is an inherently bad idea. Successful innovation is inherently about seeing which ideas are truly useful. Standardizing technologies that haven't already proven themselves in the marketplace means that bad ideas inevitably leak in. To cite my favorite example, compare the adoption and success of Spring versus CMP [container-managed persistence]. So I think the JCP needs to focus on standardizing innovations that have already happened as opposed to trying to inspire

innovation in the standards process. That's particularly true of bringing in very new technologies and adding them to the Java platform.

Now, on the flip side, the JCP process is also the mechanism by which the Java platform has evolved. And I think the more open and community-based that we can make the specifications and the implementations, the better the platform is going to be.

So in this context, the recent direction of closely tying the work in the JCP with what's going on in the OpenJDK implementation is very positive. There's a process right now of getting some of the spec work for the Java 8 JSR 337 done

on an OpenJDK mailing list—that’s a really good thing.

**Java Magazine:** Do you have any ideas about how Oracle should best negotiate the inevitable tension between running a business and leading an open source java community?

**Milinkovich:** Let me offer a lesson from the Eclipse experience. Historically, Eclipse started as an IBM project and initially, there was lots of tension between IBM and Sun Microsystems about the future of Java and how Java should be governed. And what IBM tried in part to do when they set up the Eclipse Foundation was demonstrate how they thought Java should be governed. The lesson learned was that if you truly set it free at arm's length, good things can happen. The Eclipse Foundation today has nearly 190 organizational members. There are literally thousands of products that are built on top of Eclipse. And to a considerable degree, that happened after IBM let Eclipse go and set up the Eclipse Foundation as an independent entity.

I'm not sure if Oracle could ever do that, in part because of the business legacy of Java and the way it was initially established. But the more it can make the technology open to all comers and operate through a transparent and meritocratic community-based approach, the more successful it will be.

One area of tension between the success of Java and Oracle's business

## IDEAS FIRST

“Innovating via standards is an inherently bad idea. Successful innovation is about seeing which ideas are truly useful.”



**Magazine covers that have featured Eclipse line the walls in a hallway at the Eclipse Foundation.**

needs is in the field-of-use restrictions that are holding Java back from success in the embedded and mobile spaces. At some point—I don't know when or how—Oracle is going to have to get rid of them because in today's world, if you don't have both a technology and business adoption strategy in mobile and embedded, you're dead. I don't know the solution, but Oracle will definitely need to change this in order to make Java competitive with Android, to give but one example.

**Java Magazine:** Finally, what could

OpenJDK learn from the Eclipse community?

**Milinkovich:** OpenJDK recently announced that they were going to adopt Eclipse's approach of shipping regular, predictable release trains. I think that Eclipse has shown that predictable release schedules are very helpful to the ecosystem. So that's one lesson that's been shared already.

But in comparing OpenJDK to Eclipse we need to understand that Eclipse is truly an independent organization, set up as a separate corporation with

a board of directors that oversees the entire organization, sets the budget, sets the strategy, and so forth.

OpenJDK is, like the JCP, effectively an arm of Oracle, which puts certain constraints on the way things get done. All the intellectual property flows to Oracle and, so far, the infrastructure for OpenJDK has been constrained by the ability of Oracle to put resources into improving it. So, for example, it's taken quite some time—and it's still not done—to get an open, non-Oracle bug database available for the whole OpenJDK community.

What can OpenJDK learn from Eclipse? I think that, to the degree possible, being truly vendor neutral in a level playing field across all of the companies and individuals involved is absolutely key. The degree to which OpenJDK can bring in other companies, participants, and contributors will be the best measure of success. OpenJDK has certainly done well in the last year and a half, bringing in Red Hat, IBM, SAP, and a few others as well. That's a big step forward for the OpenJDK community. [</article>](#)

**Janice J. Heiss** is the Java acquisitions editor at Oracle and a technology editor at *Java Magazine*.

**LEARN MORE**

- [Mike Milinkovich's blog](#)
- [Follow Milinkovich on Twitter](#)

# Data Quality Tools for Java



## BEFORE

john smith iii phd  
melissa data corp.  
22382 Empresa 92688  
7145895200  
john@800miAL.con

## AFTER

**Melissa Data Corp.**  
**John Smith III PhD**  
**22382 Avenida Empresa Ste 100**  
**Rancho Santa Margarita, CA 92688-2112**  
**949-589-5200**  
**John@melissadata.com**  
**Delivery Indicator: Business**

\*Highlights indicate added and/or corrected data.

**Realtime NCOA<sup>Link</sup>**  
**Change-of-Address**  
**Web Service**  
**available**

Now, finding the right data verification tools doesn't have to be so puzzling. Melissa Data offers customizable APIs, Web services and enterprise applications to match your budget and business needs. For solutions to cleanse, validate and standardize your contact data, we're ready to help you find the perfect fit.



**Request free trials at**  
**MelissaData.com/myjava or call 1-800-MELISSA**

- Global address verification for 240 countries
- Clean and validate data at point-of-entry or in batch
- Correct misspellings, missing directionals, and confirm deliverability
- Enhance addresses with County, Census, FIPS, etc.
- Append rooftop lat/long coordinates to street addresses
- Update records with USPS and Canadian change of address info

**MELISSA DATA<sup>®</sup>**

Your Partner in Data Quality

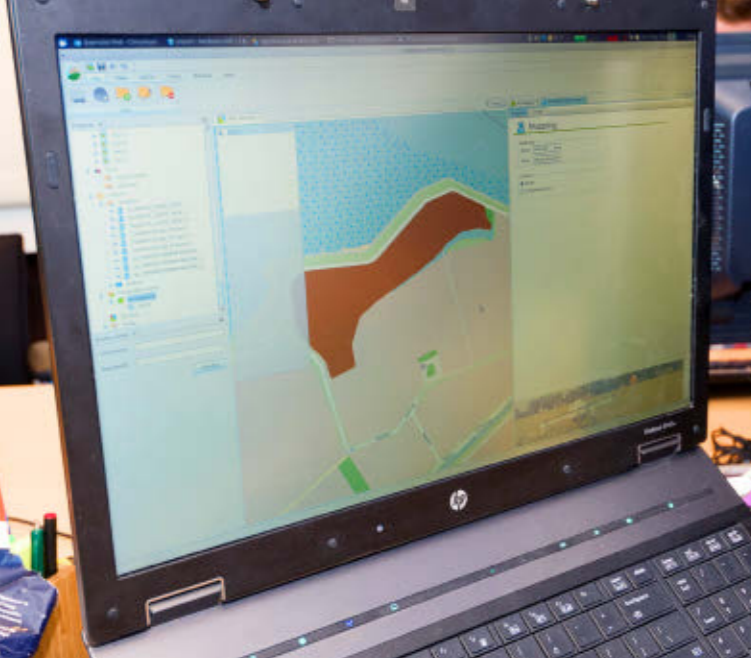


**Java version used:**  
JDK 7, NetBeans  
Platform 7,  
NetBeans IDE 7



Veenstra and members of the AgroSense Project team in the Ordina offices in Nieuwegein, the Netherlands

For too many farmers, however, all that information exists as raw data with little meaning. "Farmers have lots and lots of data, but they have little information," says Veenstra. "So much



The central component of the AgroSense platform is an onscreen map that displays not just the topography of the land, but also all the critical information that farmers need to know.

## AGRICULTURAL POWERHOUSE

The Netherlands may be a small country, but it's one of Europe's and the world's largest economies and most prosperous countries. Besides being home to large industrial organizations such as Philips Electronics NV and Unilever and oil-and-gas giant Royal Dutch Shell, the Netherlands is the third-largest exporter of agricultural products in the world, behind the United States and France. More than 56 percent of the country's arable land is under cultivation, and agriculture provides 1 in 10 jobs and generates 10 percent of the gross national product. Its leading crops are tomatoes, cucumbers, chilies, and fresh-cut plants and flowers—the country has of

course been famous for its tulips for centuries.

"Because the country is so densely populated, agricultural land is very scarce and very expensive," says Veenstra. "Farmers are sitting on millions of euros of land value, so they are eager to get the most value in crops from their land."

This has also made Dutch farmers open to a rela-

so that a lot of that data gets thrown away because they lack a platform to process that data into information."

Providing a suitable platform for just that task is the goal of the AgroSense Project, an open source farm management system being built on Java and the NetBeans Platform. Veenstra launched AgroSense—originally named “Cloudfarming”—in June 2011 with initial support and funding from [Ordina](#), an IT consulting and outsourcing firm based in Nieuwegein, the Netherlands, where he works as a software architect and developer. Although Veenstra is focused on agriculture in the Netherlands right now, he says he sees no reason why the capabilities he envisions for the AgroSense platform would not be applicable to farmers around the world.

"The goal is to make everything a farmer does available on the AgroSense platform," says Veenstra.

**BIG BUSINESS**  
Farms cover more than 56 percent of the arable land in the Netherlands and employ 1 in 10 workers.

# Precision Agriculture

*Precision agriculture*, also called *precision farming*, is a relatively new farm management concept that many see as the future of sustainable farming.

Precision agriculture differs from traditional farming practices in that it enables farmers to manage their fields at an intrafield level. In traditional agriculture, a farmer would just take a basic value of fertilizer or pesticide, for instance, and apply it to all of the land. “This approach will always be too much, especially for pesticides,” explains Timon Veenstra, project leader of the AgroSense Project, an open source software platform for precision agriculture. “While there is disease on the land, it’s usually just on one part of the land, but farmers would apply the pesticides to their entire land.”

That's not just a waste of expensive resources and pesticides; it's also potentially damaging to the land and the environment, notes Veenstra.

And too much fertilizer can result in poor-quality crops. "If you apply too much fertilizer, the crops will grow too fast and be too weak, so when the winter winds come they'll fall down," he says.

But with precision agriculture, Veenstra continues, “farmers use information from sensor values and soil samples taken from each part of the field and only need to spray pesticides on the affected areas and to apply fertilizer only to the part of the land that needs it.”

Although precision agriculture originated in the United States in the 1980s, interest has spread worldwide, including to the Netherlands, which is the world's third-largest exporter of agricultural products, behind the United States and France.

## What's Driving Open Source?



Simon Phipps, president of the Open Source Initiative, talks with Java Magazine's Tori Wieldt about what's driving open source today.

tively new concept in farm management called *precision agriculture*. Also known as *precision farming*, it combines crop science, responsible environmental stewardship, and efficient farm practices so that farmers can grow more crops of higher quality while consuming fewer resources and minimizing environmental impact and footprint (see sidebar, "Precision Agriculture").

"With precision agriculture, farmers can manage the land down to the square-meter level, rather than field by field," explains Veenstra. "Precision agriculture makes farmers more productive, enabling them to produce higher quality and a higher quantity of crops."

AgroSense—a [2012 Duke's Choice Award winner](#)—is a modular farm management system for precision agriculture. The central component of the AgroSense platform is an onscreen map that displays not just the topography of the land, but also all the critical information that farmers need to know, such as soil ingredients, crop types, fertilizers and pesticides used, and much more. AgroSense uses satellite images, JPEGs, and open source and proprietary mapping services to create maps, and even allows farmers to draw their own.

The information is displayed in the context of whichever module or application is being used, explains Veenstra. "If you're operating in the fertilizer module, for example, when farmers

click on the field they will see all the information about fertilizers being used on that field, how much is being used, even if the fertilizer is in stock," he says. "And if it's not in stock and they need more, the platform will connect farmers to an online catalog so they can order it."

"When the farmer changes the context to do something else—say, plan a new crop rotation—then the entire context will switch to that new module," he adds, "and it will display the information and the appropriate tasks."

In its first release, slated for late 2012, the AgroSense platform will focus on crop planning. "With the tools

in the first release, farmers will be able to plan their crops for a year," says Veenstra. "They will be able to import a shapefile with the digital representation of their fields, or they can draw new fields on their map. Then they can select the crop they want to grow and which periods they plan to grow it in."

This first release will be geared toward the farm management home office, which usually has good internet connections, says Veenstra. In later releases, Veenstra envisions that farm workers on tractors will have a variety of mobile devices, such as cell phones and tablets, either mounted on tractors or carried in their pockets, that will



Veenstra and AgroSense Project team members catch up over coffee.

## COMMUNITY

# JAVA IN ACTION

JAVA TECH

## ABOUT US

0

f

ava  
net

olog



24

**Philip J. Gill** is a San Diego, California–based freelance writer and editor who has been following Java for 20 years.









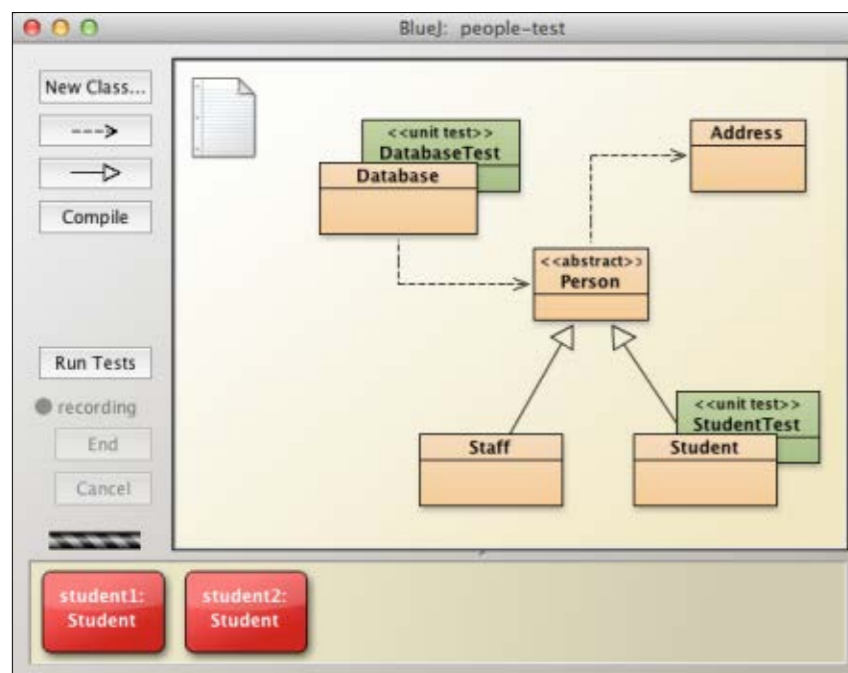






reference class; it is shown in the diagram in green. When we drag our classes in the diagram, the test class will always be kept close to the reference class. **Figure 5** shows our project with two test classes created: one for the **Student** class and one for the **Database** class.

We can now open the editor for the test class, and we see that it contains an automatically generated skeleton for a standard JUnit class. If we are familiar with JUnit, we could now start writing JUnit test cases in the standard JUnit way. This is a big improvement over repeated interactive testing, and it is very useful. However, it gets *really* interesting when we start combining interactive testing with JUnit.



### Figure 5

## Test Recording

In BlueJ, you not only have a choice between interactive testing and writing JUnit code, but you can combine both. You can record your interactive tests to get them translated automatically into replayable JUnit code. Let's play this through by creating a test case for the bug with the incorrect account name, which we discovered above.

From the test class's pop-up menu, choose **Create Test Method**, as shown in **Figure 6**. BlueJ will ask you for a name for the new test method; let's call it **testAccountName**.

Once we have selected this name, we enter the *test recording* mode. An indicator on the left shows that we are now recording, and buttons are provided to end or cancel the recording. We can now interactively carry out our test. Create a new **Student** object named James Duckling with student ID 123456. (You will also have to specify a birth date, but this does not matter much for our test.) Once the object appears

on the object bench, call its `getAccountName` method.

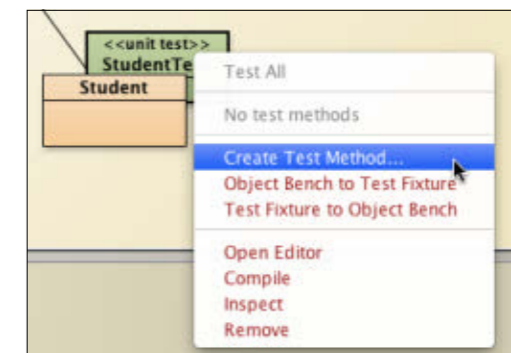
As before, we see the method result showing the (incorrect) account name, but when we're recording tests, there is an additional element in the result dialog box: an area to add an assertion. Here, we can now add the assertion for the expected result (`jam1234`; see **Figure 7**).

Once done, close the dialog box and end the recording by clicking the **End** button under the recording indicator. BlueJ will immediately generate JUnit test code for this test and add it to the `StudentTest` class. You can open the editor to inspect it.

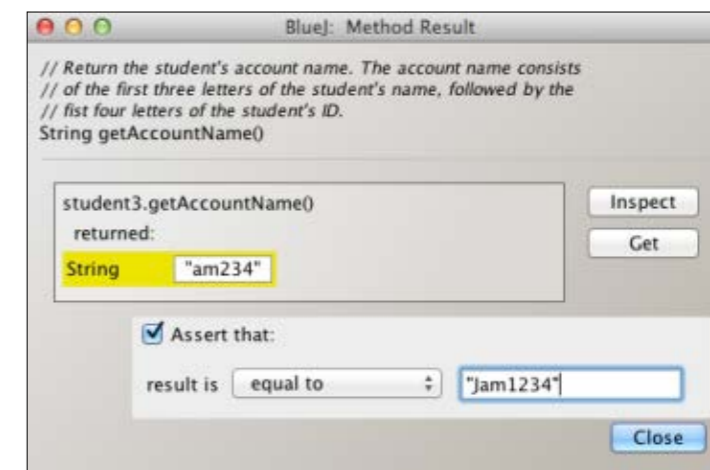
## Playback

Once we have some unit tests recorded, we will naturally want to execute our tests at some point. There are three ways to do this:

- We can select an individual test method (for example, `testAccountName`) from the `StudentTest` class's pop-up menu to run only that test method.
- We can select **Test All** from the `StudentTest` class's pop-up menu to run all tests defined in that class.



### Figure 6



### Figure 7

- We can use the **Run Tests** button in the left toolbar to run all tests in our project.
- In each case, the results of the test runs are displayed (see **Figure 8**), and if an error was detected, BlueJ can take you straight to the JUnit source where the failure originated.

## Test Fixtures

BlueJ can also use JUnit test fixtures. Just create your fixture objects interactively on the object bench as you would like them to be for the beginning of your





versions of idioms such as *filter*, *map*, and *reduce* (which we will be meeting shortly).

## Lambda Expressions in a Nutshell

In the example of different capitalization of strings, what we want to do is pass a representation of `toLowerCase()`. In order to pass this code into a version of the `contains()` method as a second parameter, we'd need to do the following:

- Find some way to treat a code snippet as though it were a value (some sort of object)
- Find a way to be able to put that block of code into a variable

In other words, we want to wrap up bits of logic in something that can then be passed around. To make this a little more concrete, let's look at a couple of basic examples of lambda expressions that can be used to replace existing java code.

**Filtering.** A good example of a snippet of code that you might want to pass around is a filter. For example, imagine you were using (a pre-Java SE 7) [java.io.FileFilter](#) in order to identify directories that belong inside a given path, as shown in **Listing 1**. With a lambda expression, this can be dramatically simplified, as shown in **Listing 2**.

The type (**FileFilter**) is inferred

from the left side of the assignment. The right side looks like a very shortened version of the `accept()` method in the `FileFilter` interface, that is, it accepts a `File` and returns the Boolean from evaluating `f.isDirectory()`.

In fact, we can actually simplify this further due to lambda expressions taking advantage of type inference, which works as follows. The compiler knows that `FileFilter` has only one method, `accept()`, so this must be the implementation code for that method. It also knows that `accept()` takes only one parameter, which is of type `File`. Therefore, `f` must be of type `File`. See **Listing 3**. As you can see, the use of a lambda expression radically reduces the amount of boilerplate.

Once you're used to lambda expressions, they make the flow of logic much easier to read. One of the key ways this is achieved is by locating the filtering logic next to the method that wants to use it.

**Event handlers.** Another area that features anonymous inner classes quite heavily is UI code. Let's take the assignment of a click listener to a button, as shown in **Listing 4**.

This is a lot of code just to say “when the button is pressed, call this method.” Using lambda expressions, it’s possible to use the code shown in **Listing 5**.

## LISTING 1

## LISTING 2

### LISTING 3

## LISTING 4

## LISTING 5

## LISTING 6

```
File dir = new File("/an/interesting/location/");
FileFilter directoryFilter = new FileFilter() {
    public boolean accept(File file) {
        return file.isDirectory();
    }
};
File[] directories = dir.listFiles(directoryFilter);
```



[Download all listings in this issue as text](#)

The `listener` object can be reused if necessary, but if it's used only once, the code in **Listing 6** is considered good style.

In this example, the slightly odd syntax involving extra curly brackets is needed because `actionPerformed` returns `void`. We'll see more about that later.

Let's move on to take a look at the role that lambda expressions will play in writing modern code for handling collections, in particular the transition between two programming styles known as *external* and *internal iteration*.

## External Versus Internal Iteration

Up until now, the standard way of dealing with a Java collection is via external iteration. It's called *external iteration* because there is control flow, external to the collection, which is used to iterate over the elements contained in

the collection. This is the traditional way of handling collections, with which most Java programmers are very familiar—even if they don't know or use the term.

For example, language constructs such as the enhanced `for` loop create an external iterator and use that object to traverse the collection shown in **Listing 7**.

With this approach, a collection class represents a “monolithic” view of all the elements in the collection, and the collection object can provide random access to any element of the collection that the programmer might require.

In this view of the world, we iterate by calling a method—`iterator()`—on the collection instance. This returns an object of type `Iterator`, which is a more restricted view on the same collection. It does not expose an interface allowing random access; instead, it is designed









means that you can safely create JavaFX objects of any type and add them to the scene graph even from within your Spring configuration.

In this particular example, we load a subconfiguration class called `ScreensConfiguration`, set the stage on that configuration object, and call a method to show the login dialog box.

## Modularizing Your UI With DI

Now that you are able to load Spring, it is time to take advantage of some of its features to help modularize your JavaFX application. The finished Customer Data App will include the following screens:

- A login screen
- A data screen
- An error screen
- An “add customer” screen

Each of these screens will be implemented as a separate JavaFX class or FXML file, and they will require references to other screens (when navigation is required) and a model (when data access or updates occur). We also want the screens to get created on first use, but we want to have no more than a single instance of each screen at a given time.

## Listings 2a and 2b

([ScreensConfiguration.java](#)) show a Spring configuration class that defines each of our screens as a separate bean.

In the `ScreensConfiguration` file, we define each screen and associated controller as a separate bean by using the `@Bean` annotation. Also, some of the beans (in our case, all that display dialog boxes) are defined with `@Scope("prototype")` so that a new instance gets created each time we fetch one. However, the main data screen gets the default scope of `"singleton"`, which means we will have only one main screen for the entire application no matter how many times the method is called.

Note that the entire configuration class is annotated with `@Lazy`, to ensure that beans will not get created until the first time they are accessed. However, this alone is not enough to ensure that beans do not get created on startup, because direct links between screens will cause a domino effect where loading the first screen loads every other referenced screen.

To avoid loading all the screens at once, rather than injecting a reference to the screens we want to connect, we instead pass in a reference to the `ScreensConfiguration` class. This still gives us the flexibility to swap implementations by providing a different `ScreensConfiguration` instance, but it defers object construction until the method for loading a specific screen gets called.

## LISTING 2a

## LISTING 2b

```
@Configuration
@Lazy
public class ScreensConfiguration {
    private Stage primaryStage;

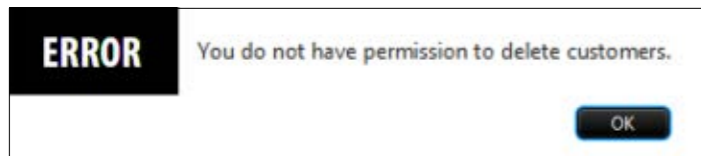
    public void setPrimaryStage(Stage primaryStage) {
        this.primaryStage = primaryStage;
    }

    public void showScreen(Parent screen) {
        primaryStage.setScene(new Scene(screen, 777, 500));
        primaryStage.show();
    }

    @Bean
    CustomerDataScreen customerDataScreen() {
        return new CustomerDataScreen(customerDataScreenController());
    }

    @Bean
    CustomerDataScreenController customerDataScreenController() {
        return new CustomerDataScreenController(this);
    }

    @Bean
    @Scope("prototype")
    FXMLDialog errorDialog() {
        return new FXMLDialog(errorController(),
            getClass().getResource("Error.fxml"), primaryStage,
            StageStyle.UNDECORATED);
    }
}
```



## Figure 2

The rest of the configuration for the model and Web services is defined in a separate class file that loads the `ScreensConfiguration` via annotation, as shown in **Listing 3** (`CustomerAppConfiguration.java`).

## Injecting Controllers into FXML

In the previous section, we defined several screens, some as Java class files and others as FXML. While it is fairly straightforward to attach a Spring-initialized controller to a Java object, doing the same for FXML was not even possible with the JavaFX 2.0 release. Fortunately, the JavaFX team added an API in the 2.1 release to specifically address this for IoC frameworks such as Contexts and Dependency Injection (CDI), Guice, and, of course, Spring.

As an example, let's take a closer look at the definition of the `ErrorDialog`. The view and controller are both created in the `ScreensConfiguration` class with the lines shown in **Listing 4**.

Notice that the `errorController` is injected as a dependency into the `FXMLDialog` class. This is a custom class I put together to encapsu-

late the logic for loading an FXML file as well as injecting the controller object. The full source code for the `FXMLDialog` is shown in **Listing 5** (`FXMLDialog.java`).

This class generalizes some of the code you would normally need in order to load an FXML file as a new dialog (Stage). However, the important part for injecting the controller is the call to `loader.setControllerFactory()`, which creates an inner class that will return the controller passed in as a parameter.

By using this technique to create and inject the controller, it is treated the same as any other Spring-managed object, letting you use DI, autowiring, and other features of the framework. The full code for the `ErrorController` is shown in **Listing 6** (`ErrorController.java`). And the base `DialogController` interface is shown in **Listing 7** (`DialogController.java`).

The FXML UI was created using **SceneBuilder**, which is a great visual tool for learning the JavaFX components and quickly mocking up UIs. The finished error dialog box I created looks like **Figure 2**.

## Conclusion

In this article, I have given you an application template for using Spring and JavaFX together that

LISTING 3 / LISTING 4 / LISTING 5 / LISTING 6 / LISTING 7

```
@Configuration
@Import(ScreensConfiguration.class)
@ImportResource("classpath:applicationContext-security.xml")
public class CustomerAppConfiguration {
    @Bean
    CustomerModel customerModel() throws IOException {
        CustomerModel customerModel = new CustomerModel();
        customerModel.setRestTemplate(restTemplate());
        customerModel.loadData();
        return customerModel;
    }

    @Bean
    RestTemplate restTemplate() {
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.setMessageConverters
            Collections.<HttpMessageConverter<?>>singletonList(
                new MappingJacksonHttpMessageConverter());
        return restTemplate;
    }
}
```



**Download all listings in this issue as text**

you can reuse in your own projects. Just taking advantage of Spring configuration and DI already gives your application design an advantage over writing in pure Java.

In Part 2, I will build out the rest of the application including

a lightweight back end with reusable model objects and one-line Web services on the client using [RestTemplate](#). </article>

**LEARN MORE**

- [Stephen Chin's blog](#)

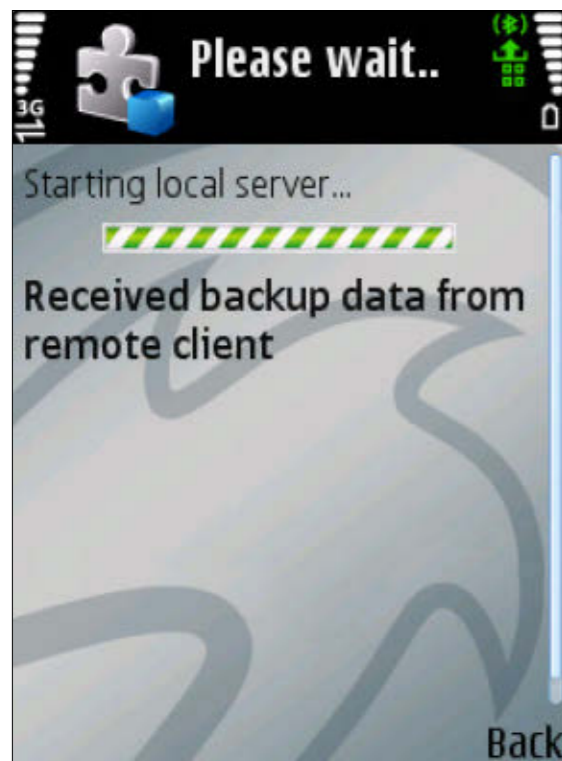




is shown in **Listings 4a** and **4b** and uses the [BlueCove](#) library, which is a JSR 82 implementation.

Try as I might, I wasn't able to get the right service code to identify my phone. The service was discovered but the name wasn't always the same as what I was expecting, so I had to hack this by identifying the correct service by the presence of the number 5. This number might be different for you.

To run this code on your desktop, you will need to have the BlueCove directory in your class path and the data.gz file must be present in the same directory where you are running this code. The combination of fulfilling these two requirements



**Figure 3**

in parallel allows the data.gz file to be sent to the phone (see **Figure 3**).

### Decompressing the Data

Now that we have received the data from the remote backup computer, the rest of the steps are easier.

In order to decompress the data, we simply pass it through the LZC decompressor. The [receiveBackup\(byte\[\] data\)](#) method in the MIDlet application is called by the [RequestHandler](#) method once the data has been successfully received. The first step in this method is the decompression of the data:

```
LZCInputStream lzc =
    new LZCInputStream(
        new ByteArrayInputStream(data));
```

### Adding the Individual Items to the User's Address Book

The Personal Information Management (PIM) API helps us in this step, although only to an extent. It provides the [PIM.getInstance\(\).fromSerialFormat\(\)](#) method (see **Listing 5**), which reads an input stream to decipher the contents and then automatically creates the right PIM item from the contents.

This is where the usefulness of this method ends. Unfortunately, it reads only the first entry in the

#### LISTING 4a

#### LISTING 4b

#### LISTING 5

```
public void servicesDiscovered(int transID,
ServiceRecord[] servRecord) {
    Operation op = null;
    OutputStream out = null;
    for (int i = 0; i < servRecord.length; i++) {
        String connURL = servRecord[i].getConnectionURL(
            ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
        System.err.println(connURL);
        // found the right service?
        if (connURL.contains("5")) {
            System.err.println("Found service, connecting to it...");
            try {
                // read the local backup file
                InputStream input =
                    new BufferedInputStream(new FileInputStream("data.gz"));
                // read it in to our bucket
                byte[] bucket = new byte[32 * 1024];
                ByteArrayOutputStream result = null;
                result = new ByteArrayOutputStream(bucket.length);
                int bytesRead = 0;
                while (bytesRead != -1) {
                    bytesRead = input.read(bucket);
                    if (bytesRead > 0) {
                        result.write(bucket, 0, bytesRead);
                    }
                }
            }
        }
    }
}
```



[Download all listings in this issue as text](#)

file; as soon as it encounters the end of this entry (by means of the END vCARD), it stops. It leaves the [InputStream](#) object open and leaves its character position on the character after the end of the entry (if there are no errors). Thus, this method allows you to read only a

single entry, which is not ideal if you are trying to recover multiple entries from the backup file.

It is easy to overcome this, however, because the method hints that you can iterate over the [InputStream](#) to get the rest of the entries. I will leave this as an



# JAVA TECH

## ABOUT US



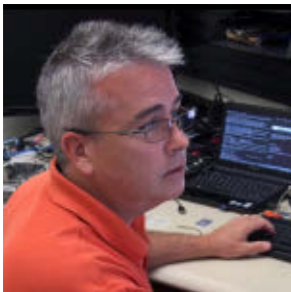
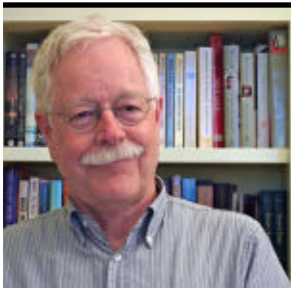
blog



## 45

- 10

45



**BILL** COURINGTON  
AND **GARY** COLLINS



# Get Started with Java SE for Embedded Devices on Raspberry Pi

# How to get Linux and Java SE for Embedded Devices running on the Raspberry Pi in less than an hour.

**I**t is hardly larger than a credit card. It costs about the same as a book on Java programming.

It's the Raspberry Pi computer, and it can support a full Java SE runtime for headless embedded applications. This article gets you started with Java on the Raspberry Pi.

**Note:** There are many potential variations of the instruc-

tions given here, especially with respect to which operations you execute on a host (desktop or laptop) computer and which you execute on the Raspberry Pi. Use your knowledge and preferences (and perhaps some of the topics in the “Performing Optional Linux Tuning and Tweaking” section of this article) to create a workflow that works for you.

# The Raspberry Pi

The Raspberry Pi is a small, low-power board built around a 700 MHz ARMv6 CPU with hardware floating point and a graphics processor integrated in a single chip. The graphics processor and the CPU share 256 MB of RAM. There are connections for USB, Ethernet, high-definition graphics, audio, an SD card, and general-purpose I/O.

**Figure 1** shows a Raspberry Pi model B with an SD card inserted for scale. The SD card plays a special role: the Raspberry Pi boots from it. This article largely consists of instructions for creating and modifying files on a bootable SD card.

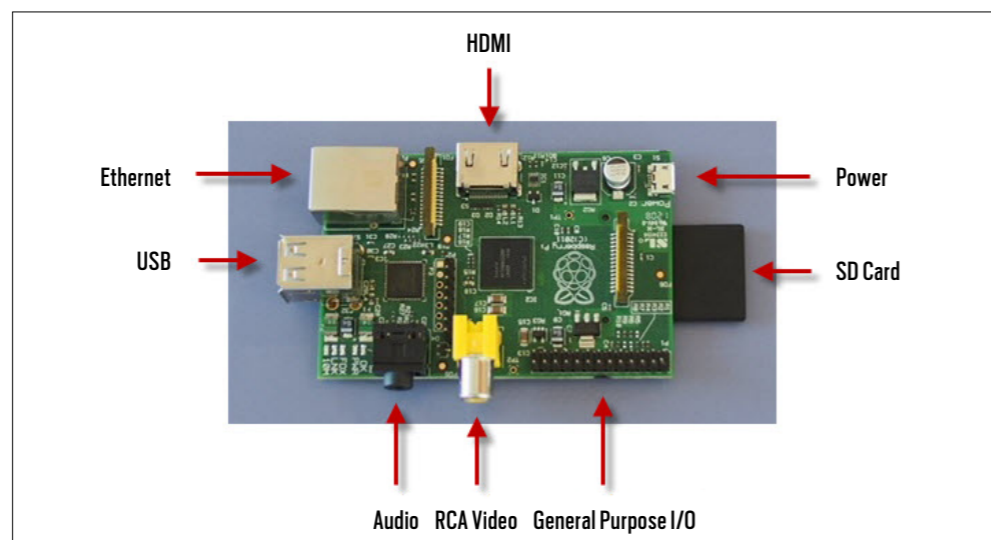
You can interact with the Raspberry Pi using a USB keyboard and mouse and an HDMI monitor or television (up to 1,080 p). The board's

Ethernet connection gives another option: interacting from a networked host via [ssh](#). The “Performing Optional Linux Tuning and Tweaking” section of this article describes how to enable [ssh](#) logins.

## Prerequisites

To set up the Raspberry Pi model B to run Java SE for Embedded Devices, you need

- A wired network to which you can connect the Raspberry Pi by an Ethernet cable. The network must supply DHCP IP addresses.
- A Linux host computer on the same wired Ethernet network as the Raspberry Pi. You can perform equivalent operations on a Microsoft Windows or Apple Mac host, but the details differ and this



### Figure 1

article does not describe them. You need the root password for the host computer. On the host you also need the following:

- A way to download a file from a Website, such as a Web browser or the **wget** utility.
- A disk partitioning tool. We use GParted here, which is simple and effective. If you don't have it, you can acquire and install it on the host as follows:

```
■ sudo apt-get install gparted
```

- A Web browser and e-mail client for downloading Java SE for Embedded Devices from Oracle.
  - An SD card reader/writer. If your host is running Linux in a virtual machine, be sure it can read from and write to SD cards.
  - A monitor that can connect directly or through an adapter to the Raspberry Pi's HDMI port.
  - A USB keyboard and mouse connected to the Raspberry Pi by a powered USB hub.
  - An SD card to hold the Raspberry Pi's operating system and Java runtime. A 4 GB card has adequate capacity for Linux, Java SE for Embedded Devices, and most embedded applications.
- Note:** Not all SD cards work with the Raspberry Pi. High-

speed cards can be too fast for the Raspberry Pi bus. We have been successful with Transcend 4 GB Micro and Patriot 4 GB Class 4 cards. The [RPi VerifiedPeripherals page](#) has lists of cards that have and have not worked for Raspberry Pi users.

- A 5 V DC micro-USB power supply capable of at least 700 mA. As mentioned on the Raspberry Pi Website, do *not* use a USB hub or a computer as a power source.

## Performing the Essential Linux Setup

When power is applied, the Raspberry Pi firmware boots from the SD card slot. Java SE for Embedded Devices runs on Linux; therefore, your first task is to get Linux on an SD card.

## Download the Debian Squeeze Linux image for the Raspberry Pi.

**Note:** These instructions are for Debian Squeeze. Other Linux implementations for the Raspberry Pi might work, but they *must* have been built for the ARM chip's software floating point capability. Images referenced by the Raspberry Pi Website might have been built for hardware floating point capability; they will *not* work with Java SE for Embedded Devices.

## LISTING 1

```
[ 110.084625] sdb: detected capacity change from 0 to
3963617280
[ 118.055249] sd 2:0:0:0: [sdb] 7741440 512-byte logical
blocks: (3.96 GB/3.69 GiB)
[ 118.059409] sd 2:0:0:0: [sdb] Assuming drive cache:
write through
[ 118.064547] sd 2:0:0:0: [sdb] Assuming drive cache:
write through
[ 118.066015] sdb: sdb1
```



[Download all listings in this issue as text](#)

1. Download the Debian Squeeze Linux image to a new directory called `~/RaspberryPi`. You can find a link to a zipped Debian Squeeze image for the Raspberry Pi [here](#). The file size is about 450 MB. To verify the downloaded file, follow the SHA-1 Checksum instructions on the page that is displayed when the download starts. The following instructions assume you have downloaded the image to a new directory `~/RaspberryPi`.
2. Unzip the image as follows:

```
$ cd ~/RaspberryPi
$ unzip *.zip
```

Unzipping creates a subdirectory called something like `debian6-19-04-2012`.

## Copy the Debian Squeeze image to the SD card.

1. On your host computer, discover the SD card device handle as follows (or use a different way if you prefer):
  - a. In a Linux terminal window, run `dmesg | tail`, which shows messages associated with device mounts. Only the final messages are of interest.
  - b. Insert the SD card into the reader/writer.
  - c. In a second terminal window, run `dmesg | tail` again. The additional lines in the output (compared to the first window) relate to the newly mounted SD card. They will vary in detail but should look something like the output shown in **Listing 1** (some lines have been broken to fit the available space).

The clue is `sdb`, which means that device handle `/dev/sdb`. `sdb1` refers to the example card's single partition. Your card might have multiple partitions.

**Note:** In the following instructions, we assume that the SD card device handle is `/dev/sdb` and that it has a single partition, `sdb1`. Adjust the instructions, if necessary, for your environment.

- 2. Unmount the SD card partition(s):  
  
■ `$ sudo umount /dev/sdb1`
- 3. Copy the Debian image to the SD card using the `dd` com-

mand, as shown in **Listing 2**.  
**Caution:** In the following command, be *certain* to enter the output file (`of`) argument value correctly. You are about to invoke a low-level disk copy that overwrites all data. Specifying the wrong device will destroy data. Using the `dd` command with an SD card is a slow operation. It can take 5 to 10 minutes for a 4 GB Class 4 card. Also, `dd` gives no indication of progress, so be patient while it slowly does its work.

- 4. If you want to be sure now that your SD card's Linux installation works, skip to the "Perform the first boot"

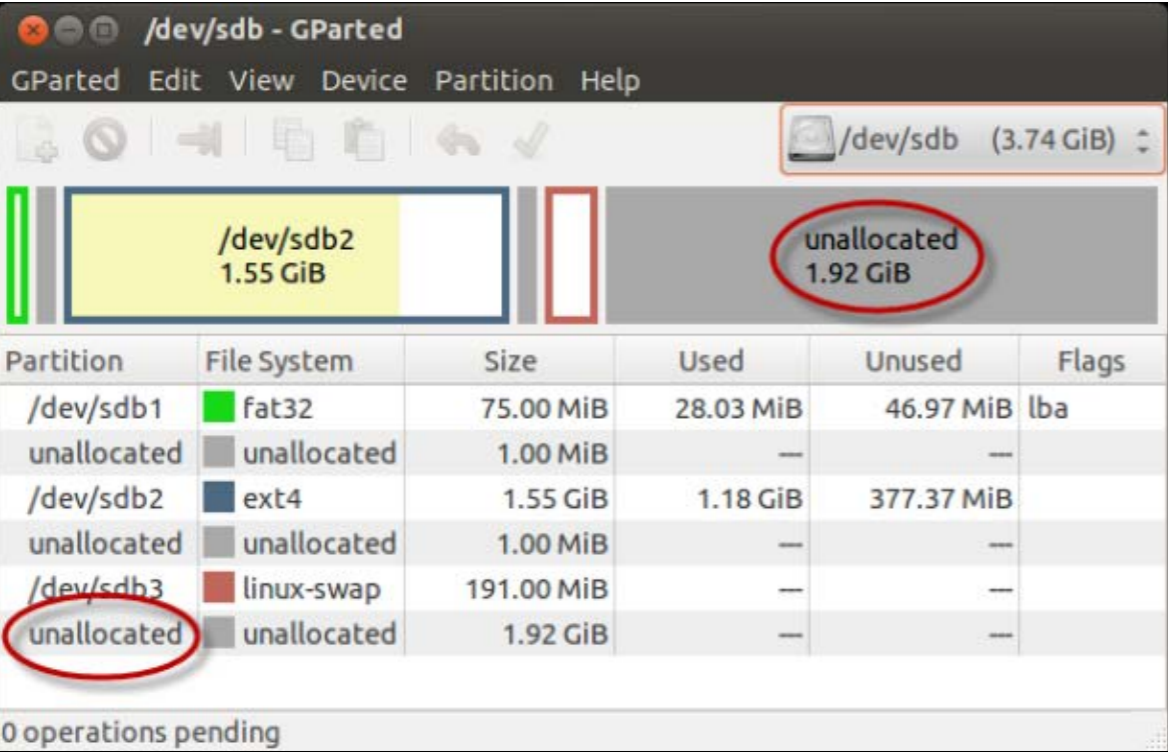


Figure 2

LISTING 2

```
$ cd ~/RaspberryPi/debian6-19-04-2012
$ sudo dd if=yourDebian.img of=/dev/sdb bs=1M
1859+1 records in
1859+1 records out
1950000000 bytes (2.0 GB) copied, 212.344 s, 9.2MB/s
```

Download all listings in this issue as text

section, and then return to the "Resize the SD card partitions" section.

**Resize the SD card partitions.** The SD card now contains three partitions and unallocated space. Now we'll add the unallocated space to the Linux file system to make more room for Java SE for Embedded Devices and applications. To assign unallocated space to the file system, we modify the partitions on the SD card by deleting and re-creating one of the partitions and resizing another.

- 1. Start GParted:

■ `$ sudo gparted`

GParted initially shows the partitions of the host's disk. You do *not* want to change these partitions.

- 2. Choose **GParted** -> **Devices** to display the partitions and unallocated space on `/dev/sdb`, as shown in **Figure 2**. The three partitions are

- `/dev/sdb1`: Boot partition.
  - `/dev/sdb2`: Linux root file system. Most of its 1.5 GB allocation is already used.
  - `/dev/sdb3`: Linux swap area. About half the card's space is unallocated. The biggest chunk of free space follows the `/dev/sdb3` partition. To add this space to `/dev/sdb2` (the file system), we temporarily delete `/dev/sdb3`, extend `/dev/sdb2`, and then re-create `/dev/sdb3`.
- 3. In the Partition column, select `/dev/sdb3` (linux-swap); then right-click and choose **Delete**. The partition is shown as deleted (see **Figure 3**), but GParted has actually queued the operation and will run it later.
  - 4. In the Partition column, select `/dev/sdb2`, which is the Linux file system; then right-click and choose **Resize/Move**.

5. In the dialog box that appears, drag the right arrow until **Free space following (MiB)** is about 512 (or however much you want to leave for

swap), as shown in **Figure 4**. You might not be able to drag to exactly 512, but the exact size is not important.
6. Click **Resize/Move** to queue

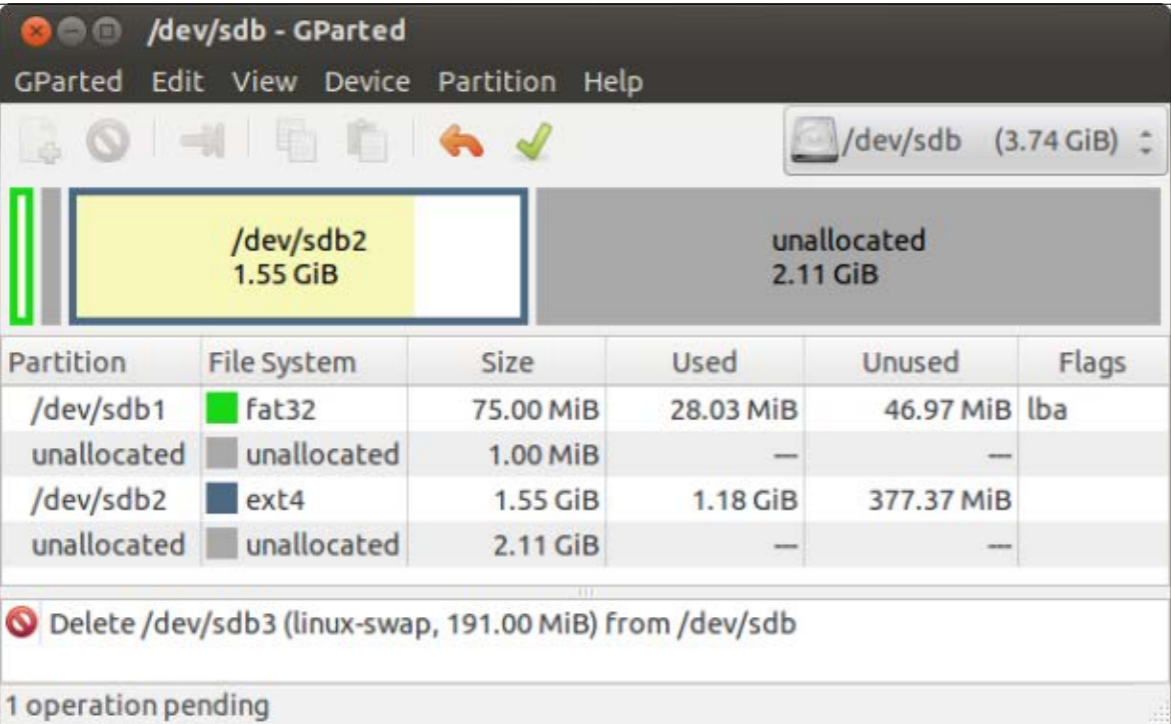


Figure 3

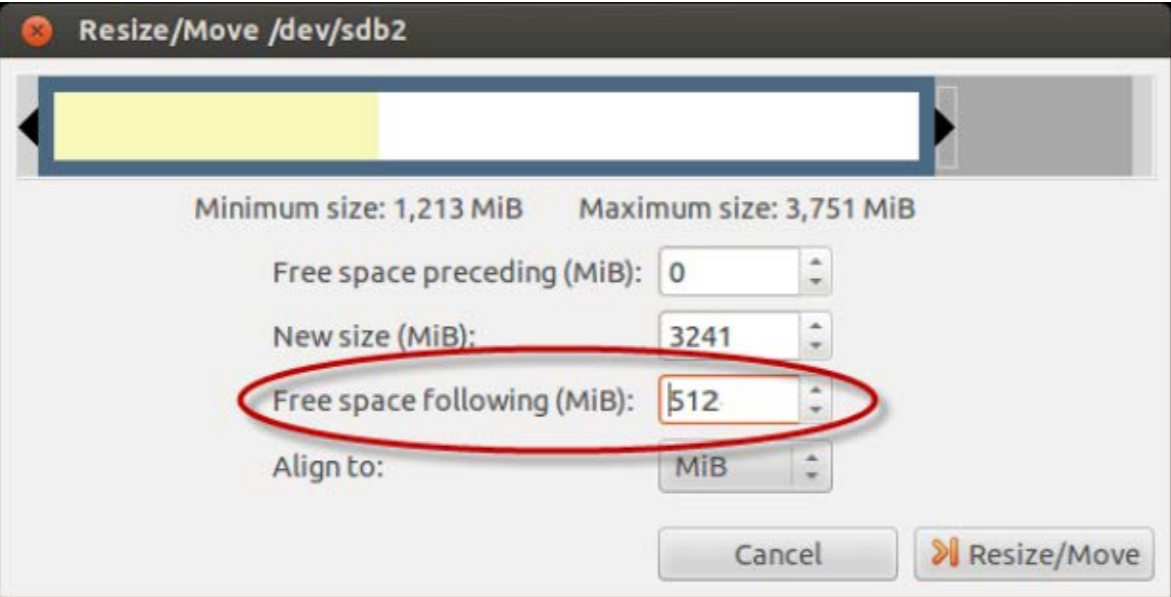


Figure 4

- the change.
7. In the “Moving a partition” warning dialog box that appears, click **OK**. Now that we’ve grown the file
- system partition, we re-create a swap partition.
8. Select the 512 MiB unallo-
- cated partition, right-click, and choose **New**.

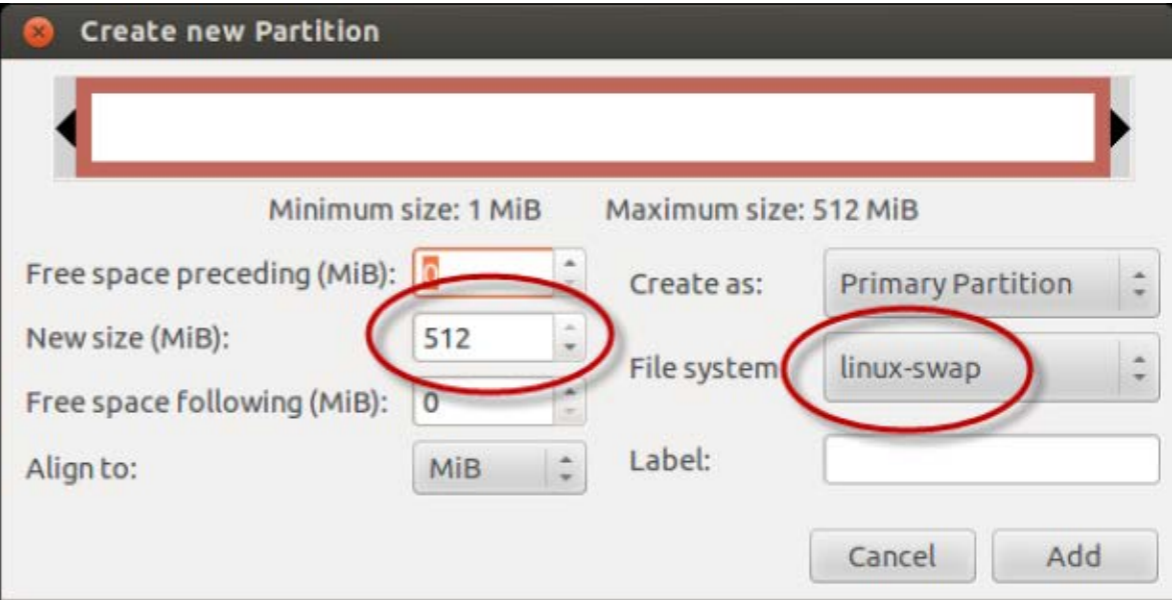


Figure 5

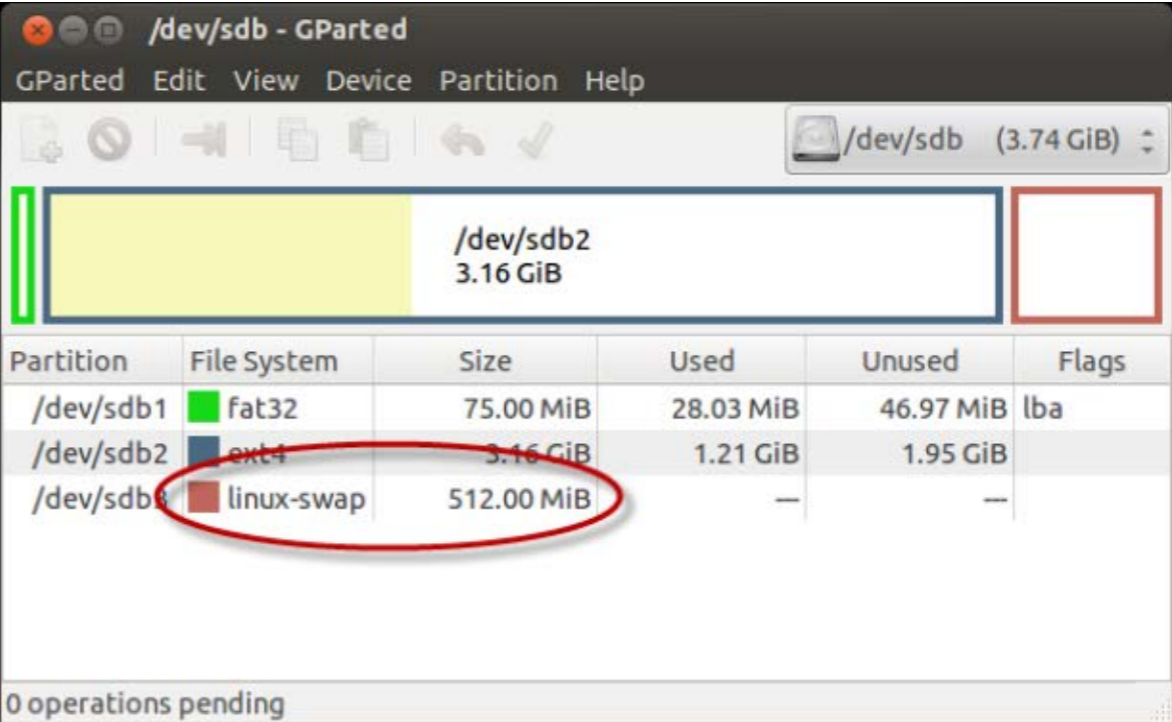


Figure 6



## Performing the Java SE for Embedded Devices Setup

Oracle offers Java SE for Embedded Devices binaries free of charge for development and testing use. In this section, we use the host to download Java SE for Embedded Devices and copy it over the network to the Raspberry Pi's Linux file system. Then, still working remotely, we unpack the download and launch Java.

## Acquire the Java SE for Embedded Devices evaluation software.

1. In the host computer's Web browser, go to the Java SE for Embedded Devices [download page](#), which is shown in **Figure 7**.  
The Web page might have changed in detail since this article was written.
2. In the Java SE Embedded 7 column, click the **EABI, VFP, SoftFB ABI, Little Endian** link.  
A survey page appears.
3. Complete the survey and click **Submit**.  
A verification page appears.
4. Examine the verification page for correctness (in particular, verify your e-mail address), and then click **Submit**.  
In a short time, an e-mail from JavaSE-Embedded-LinkNotify appears in your inbox.
5. Click the link in the e-mail.

Another download page appears, as shown in **Figure 8** (some details might have changed since this article was written):

6. Select the **Accept License Agreement** option, and then click the link in the ARMv6/7 Linux - Headless row.  
**Note:** At the time this article was written, the ARMv7 Linux - Headful software had not been tested on the Raspberry Pi. The ARMv7 Linux - Headless - Server Compiler software does not work on the Raspberry Pi, because that software requires an ARMv7 chip. If you want an ARMv7-compatible Java run-time, consider the [ARM JDK downloadable](#).
7. Save the .gz file to your Downloads directory.
8. Log in over the network to the Raspberry Pi as user **pi**, and create a directory for the java download:

```
$ ssh pi@RaspberryIP mkdir \
/home/pi/java
```

9. Copy the Java download to the new directory on the Raspberry Pi:

```
$ scp ~/Downloads/*.gz \
pi@RaspberryIP:/home/pi/java
```

### LISTING 3

```
$ ls ejre*
ejre1.7.0_04
$ cd ejre*
$ ./bin/java -version
java version "1.7.0_04-ea"
Java(TM) SE Runtime Environment for Embedded (build
1.7.0_04-ea-b20, headless)
Java Hotspot(TM) Embedded Client VM (build 23.0-b21, mixed mode)
```

 [Download all listings in this issue as text](#)

- 10.** Unpack the Java download:

```
$ ssh pi@RaspberryIP
$ cd /home/pi/java
$ tar -zxvf *.gz
...
```

- 11.** Launch Java, as shown in **Listing 3**.  
Some details, such as version and build numbers, might have changed since this article was written.
- 12.** Optionally, remove the download file:

- `$ rm /home/pi/java/*.gz`

## Performing Optional Linux Tuning and Tweaking

The procedures in this section can make your Raspberry Pi system run more reliably, more responsively, or more conveniently. Choose those you like and run them on the

Raspberry Pi. To simplify diagnosing problems, reboot after each change you make.

**Set up a static IP address.** If your network reuses dynamic IP addresses, you can skip changing to a static IP address, because the Raspberry Pi's IP address will effectively be constant. If the Raspberry Pi's dynamic IP address is likely to differ each time it is booted, a static IP address is more convenient for using [ssh](#).

In the following procedure, we assume that you want to assign the host name `raspberrypi` to the Raspberry Pi.

1. Begin by opening `/etc/hosts` as superuser in a text editor, for example, `sudo vi/etc/hosts`.
2. Add a line for the Raspberry Pi consisting of a static IP address that is valid in your network, an optional domain

name, and aliases.

**Listing 4** shows a hypothetical example in which we designate the Raspberry Pi as `raspberrypi`.

3. Save the file and exit the editor.
4. As superuser, open `/etc/hostname` in a text editor.
5. Add a line analogous to this: `raspberrypi`.
6. Save the file and exit the editor.

The following instructions can prevent a problem in which an overloaded network causes the operating system to change the Raspberry Pi's static IP address. As a result, `ssh` or other operations that use the static IP address stop working.

1. Discover the Ethernet controller's hardware address by running the command shown in **Listing 5** (some output lines have been broken to fit the text column). In **Listing 5**, each `x` in the output stands for an address digit. In this example, the name of the Ethernet controller is `eth0`. The hardware address consists of the hex digits following `link/ether`, so in this example, the address is `b8:27:eb:b5:e8:90`.
2. Now that you have the Ethernet controller's hard-

ware address, as superuser, open `/etc/network/interfaces` in a text editor and add lines analogous to those that follow the `# New entries` comment shown in **Listing 6** (substitute your network details).

**Enable swapping and optimize file system access time.** By default, swapping is disabled in Debian Linux. If an operation exceeds the Raspberry Pi's 256 KB RAM, the system crashes. Less serious is the fact that, by default, the file system spends time maintaining a last-accessed time for each file, which is of little use in embedded applications. Maintaining last-accessed times can also reduce the life of an SD card. Both default behaviors are specified in the `/etc/fstab` file.

To enable swapping, do the following.

**Note:** Swapping is slow on an SD card. For an embedded application, you probably want to manage memory so as to avoid swapping.

1. As superuser, open `/etc/fstab` in an editor such as `vi`. The file looks like **Listing 7**.
2. To enable swapping, uncomment the third line.
3. To eliminate the last-accessed overhead, insert a `p2` mount point between `p1` and `p3`. The `noatime` and `nodiratime` options do the job.

## LISTING 4

## LISTING 5

## LISTING 6

## LISTING 7

## LISTING 8

```
127.0.0.1 localhost 192.168.0.100 raspberrypi.yourDomain
raspberrypi loghost
```



[Download all listings in this issue as text](#)

When the file looks similar to **Listing 8**, save it and exit the editor.

4. Now assign `p3` (the swap partition we created on the host as `/dev/sdb3`) to the swap space:

```
$ sudo mkswap /dev/
mmcblk0p3
```

**Set the time zone and locale.** By default, the Raspberry Pi is configured for the Europe/London time zone and the `en_GB.UTF-8` locale. To localize your computer, follow these instructions.

1. To change the time zone, enter this command:

```
$ sudo dpkg-reconfigure \
tzdata
```

A pseudographical user interface appears, as shown in **Figure 9**.

2. Move the selection cursor through regions and cities using the Up and Down arrow keys. Use the Left and Right arrow keys to highlight `<Ok>` or `<Cancel>` and press **Return**.
3. Set the locale by running the following command:





ORACLE®



# JAVA TECH

## ABOUT US

For these examples, I'm using Akka 2.0.3 and Scala 2.9.2. If neither is currently on your machine, you can download the [Typesafe Stack](#) as a whole, which manages some of the dependencies

○

ava  
net

log





tional object-oriented principles, offers no encapsulation of its state, because it needs none—the whole point of the message is to convey data, not to hide it or offer any kind of processing.

From an implementation standpoint, yes, all three kinds of entities (objects, actors, and messages) are built using the same atoms (classes, fields, and methods) but in very different proportions. For example, if your actors are trying to track business data in fields, you're closer to building an object than an actor. If your messages are defining anything other than public fields (and maybe a few convenience methods such as `toString()` for easier diagnostic messages), you're closer again to building an object than a message.

And yes, all this flies directly in the face of traditional object orientation. That's a good thing.

Getting back to the example code, what we're doing here is creating an instance of an actor (`HelloActor`) and sending it a message (`Start`). This is the heart of actor-style programming: creating entities (actors) that respond to messages and, depending on the purpose of the message, replying back with values or passing those messages on to other actors.

Take careful note: At no point will one object directly invoke a

method on another—instead, message passing takes place, and any data inside those messages (of which there is none, yet) is copied instead of passed by reference.

In `main()`, we first initialize the Akka actors system by obtaining an instance of `ActorSystem`, and then we use `Props` (think of it as a dictionary of actors for now) to obtain an instance of the `HelloActor` actor.

Note that where we get this actor from is hidden from us—it could be local or remote. We don't care. (Obviously transmission times will be much longer if it's remote, but since actor-delivered messages are intended to be asynchronous, we shouldn't care).

Then, we use the `!` method on the actor to send the `Start` message to it, which tells it to print to the console.

`HelloActor` is the actor implementation, and it's pretty straightforward. As a class, it derives from the base class `Actor`, and defines a `receive` method that pattern-matches for incoming `Start` objects.

Akka can use any `Serializable` type as a message type, but the type-safe tendencies of Scala lead us to prefer using simple class types (*case class* and *case object* are perfect for this) rather than data-driven types that might be misinterpreted at runtime. For

## LISTING 2

```
object Main {
  def main(args: Array[String]): Unit = {
    val system = ActorSystem()
    system.actorOf(Props[HelloActor]) ! "Start"
  }
}

class HelloActor extends Actor {
  def receive = {
    case "start" => System.out.println("Hello, Akka!")
  }
}
```



[Download all listings in this issue as text](#)

example, we could change the code to use Strings instead, as shown in **Listing 2**.

But the dangers here are exactly what we saw in the earlier code, because now that we're using a String—and the data held inside the String—to convey the message, typos and case-sensitivity become issues. Granted, we could set a “corporate standard” that says “all messages must be in uppercase,” but this becomes one more thing we have to remember, as opposed to something the compiler can check for us.

If we go back to the original code, trying to send a `Strat` object will yield a compiler error, whereas trying to send a `Strat` string will compile and just do nothing.

(Worse yet, even if you're unit testing the `HelloActor`, the tests won't catch it—the error is in the client code, not the actor.)

### Cut! Horrible! Do It Again!

We have two problems with the code as it currently is. One is that it never terminates. The other is that it's a pretty limited example. What if we want to print more than just “Hello” or we want to repeat that message a few times, to name a few possible enhancements?

Fixing the first issue is easy: we have to tell the Akka runtime to stop. When the `HelloActor` was created, the Akka runtime spun up a collection of threads under the hood. Then, when a message was received by the Akka run-









## ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// NOVEMBER/DECEMBER 2012